REMARKS

Reconsideration and allowance of the subject application are respectfully requested. Applicant thanks the Examiner for total consideration given the present application. Claims 1-5 and 7 were pending prior to the Office Action. Claims 8-10 have been added through this reply. No claims have been canceled through this reply. Therefore, claims 1-5 and 7-10 are pending. Claims 1-2, 4-5, and 7-10 are independent. Applicant respectfully requests reconsideration of the rejected claims in light of the remarks presented herein, and earnestly seeks a timely allowance of all pending claims.

Claim Rejection - 35 U.S.C. § 103(a)

Claims 1-5 and 7 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Ko et al. (U.S. Patent 6,597,950) in view of Bates et al. (U.S. Patent Publication 2003/0041315). Applicant respectfully traverses this rejection.

For a Section 103 rejection to be proper, a *prima facie* case of obviousness must be established. *See M.P.E.P. 2142.* One requirement to establish a *prima facie* case of obviousness is that the prior art references, when combined, must teach or suggest all claim limitations. *See M.P.E.P. 2142; M.P.E.P. 706.02(j).* Thus, if the cited references fail to teach or suggest one or more elements, then the rejection is improper and must be withdrawn.

Argument A: Features of claims 1-2 and 7-8 not taught:

Independent claims 1-2 and 7 have been amended to include additional limitations. More specifically, claim 1 as amended recites, *inter alia*, "concluding that the process executed based on the instruction codes contained in the data is a malicious process, when the instruction code for calling the instruction code group for executing the predetermined process is associated with the branch destination address and the call destination address of the instruction code is between the branch origin address and the branch destination address."

The claimed invention of claims 1, 2, 7 and 8 describes a configuration of concluding that a malicious process is executed by input data, when a call instruction (*i.e.*, an instruction for

calling an instruction code group) is at a branch destination address of a jmp instruction (*i.e.*, a branch instruction) and a call destination address of the call instruction is between a branch origin address and the branch destination address of the jmp instruction.

None of the references cited in the Office Action disclose or suggest such a configuration of concluding that a malicious process is executed by input data based on the criteria as described above.

Bates merely discloses a technology for retrieving a conditional branch in a source code by use of a CDG (Control Dependence Graph), and then, setting a breakpoint with respect to the retrieved branch. However, Bates only discloses a technology for inserting a breakpoint. In this case, for example, such a branch to a subroutine from a main-routine will be detected as a breakpoint, even when the branch cannot be regarded as a malicious one. In other words, in Bates, a purpose of inserting a breakpoint is not to distinguish a malicious process from a normal process. Therefore, it is impossible to separately detect only a malicious process even based on a reference to Bates. In sum, Bates does not disclosed a configuration of concluding that a malicious process is executed by input data based on the criteria as described above.

Further, Ko can not be relied on to make up for the deficiencies of Bates. Ko discloses a technology for detecting a macro virus by statically analyzing macro operations in a document. More specifically, an analyzer 212 compares macro operations encountered during the flow analysis with suspect macro operations specified in profile database 214 to detect a macro virus (please see the lines 17-19, in column 5). These suspect macro operations specified in the profile data base 214 include operations such as modifying data within another document, modifying other files in the computer system, deleting other files in the computer system, and modifying an OS (operating system) parameters in the computer system etc.

As described above, Ko only discloses a technology for analyzing whether or not a specific macro instruction for instructing execution of modification or deletion and an object subjected to the modification or deletion (*e.g.*, data, a file, or a parameter etc.) are described in the macro operations contained in a document, with reference to the profile database 214. Ko fails to disclose or suggest the configuration of the invention peculiar to claim 1 of the present application, concluding that a malicious process is executed by input data, when a call instruction

(*i.e.*, an instruction for calling an instruction code group) is at a branch destination address of a jmp instruction (*i.e.*, a branch instruction) and a call destination address of the call instruction is between a branch origin address and the branch destination address.

Further, Baratloo can not be relied on to make up for the deficiencies of Bates and Ko. Baratloo discloses a technology for detecting an attack code which is likely to cause a buffer overflow. Further, Baratloo discloses the first method (Libsafe) of intercepting all calls to library functions that are known to have vulnerability, and the second method (Libverify) of binary re-writing of the process memory to force verification of critical elements of stacks.

In the first method described in Baratloo, a specific library called a "libsafe library" that is used to estimate a safe upper limit on the size of buffers automatically (please see Figure 4). And therefore, Baratloo fails to disclose or suggest a configuration of judging whether or not a process is a malicious one by directly analyzing the instruction structure in input data as that of the invention according to claims 1, 2, 7 and 8 of the present application.

Furthermore, the second method described in Baratloo is a method of verifying a function's return address before use, a scheme similar to that found in StackGuard. Here, the verification of the return address is performed by injecting a verification code at the start of the process execution. And therefore, Baratloo fails to disclose or suggest a configuration of judging whether or not a process is a malicious one by directly analyzing the instruction structure in input data as that of the invention according to claims 1, 2, 7 and 8 of the present application.

On the contrary, in claims 1, 2, 7 and 8, there describes a configuration of concluding that a malicious process is executed by input data, when a call instruction (*i.e.*, an instruction for calling an instruction code group) is at a branch destination address of a jmp instruction (*i.e.*, a branch instruction) and a call destination address of the call instruction is between a branch origin address and the branch destination address, as described above.

Regarding a "Shellcode", (corresponding to a malicious code described in the present Specification) in data to be used for computer viruses and cracking attacks etc., the Shellcode is injected from outside into an already loaded and executed program subjected to an attack, as a part of data to be processed by the program. Memory corruption, for example, a buffer overflow, is caused by input data because of the vulnerability of the program subjected to the attack,

resulting in executing the Shellcode in an undesirable manner against an original purpose of the program. In summary, in the *execution* of the Shellcode, the Shellcode is not loaded on a memory and executed by means of an OS, unlike a normal program. Therefore, the OS does not recognize the Shellcode injected from outside as an executable code in an object format, much less notice the presence of static data existing in the Shellcode. Namely, the Shellcode is not provided with adjustment by the OS for conforming the actual address at which static data existing in the Shellcode is arranged to the address in the instruction referring to the data, at the time that the Shellcode is injected. Here, the Shellcode should recognize the memory address (which is arranged as static data) as its own memory address by any method at the time of the execution of the Shellcode. Otherwise, the original purpose of the Shellcode cannot be achieved.

As a possible method for the Shellcode to recognize its own memory address, raised is an example of arranging a call instruction for calling a subroutine at a branch destination of a jpm instruction (a branch instruction), and obtaining a return address positioned at the top of a stack in the subroutine called by the call instruction. The call instruction has a characteristic of pushing an address of the next positioned instruction as a return address to the stack, and after that, branching to a call destination. For this characteristic, the Shellcode recognizes the first address of the instruction arranged next to the call instruction, and then, calculates the position of an address of static data existing in a distance with specified bytes from the first address by adding a relative distance to the static data to the recognized address, thereby confirming the address of the static data as an address of the static data of the Shellcode itself.

In sum, the inventions according to claims 1, 2, 7 and 8 of the present application relates to a configuration of analyzing an instruction structure contained in input data with attention to such characteristic of the call instruction and judging whether or not there is a process in which a Shellcode recognizes its own memory address, thereby judging whether or not a malicious process is executed. Then, all the references cited in the Office Action fail to describe or suggest such a configuration of concluding that a malicious process is executed by input data, when a call instruction (*i.e.*, an instruction for calling an instruction code group) is at a branch destination address of a jmp instruction (*i.e.*, a branch instruction) and a call destination address of the call instruction is between the branch origin address and the branch destination address of

the jmp instruction" to judge whether or not there is a process in which the Shellcode recognizes its own memory address, and to judge whether or not a malicious process is executed, as described in the present invention. Therefore, claims 1-2 and 7-8 as amended are submitted to be allowable over the cited prior art for at least this reason.

Dependent claim 3 is allowable for the reasons set forth above with regards to claim 2 at least based on their dependency on claim 2.

Accordingly, Applicant respectfully requests that the Examiner reconsider and withdraw the rejection of claims 1-3 and 7 under 35 U.S.C. § 103(a).

Reconsideration and allowance of claims 1-3 and 7-8 are respectfully requested for at least these reasons.

Argument B: Features of claims 4 and 9 not taught:

Independent claim 4 has been amended to include additional limitations. More specifically, claim 4 as amended recites, *inter alia*, "it is concluded that the process executed based on the instruction codes contained in the data is a malicious process, when the instruction code for calling the instruction code group for executing the predetermined process is in the data and the predetermined character string is associated with the return address of the instruction code group."

The claimed inventions according to claims 4 and 9 describe a configuration of concluding that a malicious process is executed by input data, when a call instruction (*i.e.*, an instruction code for calling an instruction code group for executing a predetermined process) is in the data and the predetermined character string is associated with the return address of the instruction code group.

None of the cited references disclose or suggest a configuration of concluding that a malicious process is executed by input data based on the criteria as described above. Moreover, as described above (in the arguments towards claims 1, 2, and 7-8), all the references cited in the Office Action fail to describe a configuration of analyzing an instruction structure contained in input data with attention to such characteristic of the call instruction and judging whether or not there is a process in which a Shellcode recognizes its own memory address, thereby judging

whether or not a malicious process is executed. Therefore, claim 4 and 9 as amended are submitted to be allowable over the cited prior art for at least this reason.

Accordingly, Applicant respectfully requests that the Examiner reconsider and withdraw the rejection of claim 4 under 35 U.S.C. § 103(a).

Reconsideration and allowance of claims 4 and 9 are respectfully requested for at least these reasons.

Argument C: Features of claims 5 and 10 not taught:

Independent claim 5 has been amended to include additional limitations. More specifically, claim 5 as amended recites, *inter alia*, "it is concluded that the process executed based on the instruction codes contained in the data is a malicious process, when the instruction code for calling the instruction code group for executing the predetermined process is in the data and the instruction code for obtaining the return address of the instruction code group is contained in the instruction code group."

The claimed invention according to claims 5 and 10 describes a configuration of concluding that a malicious process is executed by input data, when a call instruction (*i.e.*, an instruction code for calling an instruction code group for executing a predetermined process) is in the data and an instruction code (*i.e.*, a pop instruction) for obtaining the return address of the call instruction is contained in the instruction code group.

None of the cited references disclose or suggest a structure of concluding that a malicious process is executed by input data based on the criteria as described above. Moreover, as described above (in the arguments towards claims 1, 2, and 7-8), all the references cited in the Office Action fail to describe a configuration of analyzing an instruction structure contained in input data with attention to such characteristic of the call instruction and judging whether or not there is a process in which a Shellcode recognizes its own memory address, thereby judging whether or not a malicious process is executed.

Assuming arguendo, that it has been known that a call instruction for calling a subroutine pushes a memory address at which the next instruction is arranged to a stack as a return address from the subroutine, and after that, branches to a call destination address. Accordingly, the

memory address retained in the stack can be retrieved at the call destination, and then, a memory address of the instruction code prepared by an attacker can be determined relative to the memory address at which the call instruction is arranged, by adding an appropriate constant number thereto.

However, such a process of determining a memory address of any instruction code by use of the characteristics of the call instruction as described above is never used for a normal program (*i.e.*, a program which does not contain data for executing a malicious process). The reason for this lies in that if a memory address arranged on the top of a stack by a call instruction is used in a subroutine of a call destination, it is impossible to return to a return address (*i.e.*, a call origin), when a control reaches to a return instruction in the subroutine.

In the inventions according to claims 5 and 10 describe a configuration of judging whether or not there is a process in which the Shellcode recognizes its own memory address by detecting an extremely peculiar instruction structure which cannot be seen in such a normal program. Therefore, claim 5 and 10 as amended are submitted to be allowable over the cited prior art for at least this reason.

Accordingly, Applicant respectfully requests that the Examiner reconsider and withdraw the rejection of claim 5 under 35 U.S.C. § 103(a).

Reconsideration and allowance of claims 5 and 10 are respectfully requested for at least these reasons.

Conclusion

Therefore, for at least these reasons, all claims are believed to be distinguishable over the combination of Ko and Bates, individually or in any combination. It has been shown above that the cited references, individually or in combination, may not be relied upon to show at least these features. Therefore, claims 1-5 and 7-10 are distinguishable over the cited references.

In view of the above remarks and amendments, it is believed that the pending application is in condition for allowance.

Applicant respectfully requests that the the pending application be allowed.

Should there be any outstanding matters that need to be resolved in the present application, the Examiner is respectfully requested to contact Aslan Ettehadieh Reg. No. 62,278 at the telephone number of the undersigned below, to conduct an interview in an effort to expedite prosecution in connection with the present application.

If necessary, the Commissioner is hereby authorized in this, concurrent, and future replies to charge payment or credit any overpayment to Deposit Account No. 02-2448 for any additional fees required under 37.C.F.R. §§1.16 or 1.147; particularly, extension of time fees.

Dated: January 21, 2009

Respectfully submitted,

By _____
Michael K. Mutter
Registration No.: 29,680
BIRCH, STEWART, KOLASCH & BIRCH, LLP
8110 Gatehouse Road
Suite 100 East
P.O. Box 747
Falls Church, Virginia 22040-0747
(703) 205-8000
Attorney for Applicant